Proxying is Enough

Security of Proxying in TLS Oracles and AEAD Context Unforgeability

Zhongtang Luo\textsuperscript{1}, Yanxue Jia\textsuperscript{1}, Yaobin Shen\textsuperscript{2}, Aniket Kate\textsuperscript{13}
\textsuperscript{1}Purdue University, \textsuperscript{2}Xiamen University, \textsuperscript{3}Supra Research
SBC 2024
Oracles pull in information from Web2.
(e.g. exchange rate, prediction market, etc.)
(1) What is my credit score?

(2) Your credit score is 720.

How can we pull in more information?
(1) What is my credit score?

(2) Your credit score is 720.

(3) I confirm the user’s credit score is over 700.

Root of trust: TLS certificate
Caveat: TLS is a symmetric encryption!
Caveat: TLS is a symmetric encryption!

An adversarial user can produce any transcript from the key.
Caveat:

TLS is a symmetric encryption!

An adversarial user can produce any transcript from the key.

Oracle has to be involved in the communication without changing the TLS protocol.
The user reveals the needed part of the plaintext at the end (with some proof).
Big question: Is it secure?
Proxy-Based TLS Oracle

Conjectured **insecure** since DECO in 2020
- Conjectured **insecure** since DECO in 2020

- Key commitment attack: The ciphertext may decrypt to a different plaintext with a different key.
Proxy-Based TLS Oracle

Conjectured **insecure** since DECO in 2020

- Key commitment attack: The ciphertext may decrypt to a different plaintext with a different key.
- User can decrypt the same ciphertext into different plaintexts with different keys.
Proxy-Based TLS Oracle

- Conjectured **insecure** since DECO in 2020
- Key commitment attack: The ciphertext may decrypt to a different plaintext with a different key.
- User can decrypt the same ciphertext into different plaintexts with different keys.
- A whole plethora of work on ensuring key commitment:
  - **DECO**: Liberating Web Data Using Decentralized Oracles for TLS
  - **DIDO**: Data Provenance from Restricted TLS 1.3 Websites
  - **Janus**: Fast Privacy-Preserving Data Provenance for TLS
  - **Lightweight Authentication of Web Data via Garble-Then-Prove
  - **ORIGO**: Proving Provenance of Sensitive Data with Constant Communication
  - ...
But is it really insecure?
Popular fix on key commitment: **Padding**
\[1\]
i.e. add 128 bytes of 0s to the front of the plaintext.

\[1\]https://eprint.iacr.org/2020/1456
Entering the Cryptography Territory...

- Popular fix on key commitment: **Padding**\(^1\)
i.e. add 128 bytes of 0s to the front of the plaintext.
- Rationale: Hard to decrypt the same ciphertext to the same plaintext (0s) with different keys.

\(^1\)https://eprint.iacr.org/2020/1456
Proxying is Enough: Security of Proxying in TLS Oracles and AEAD Context Unforgeability
Hard to decrypt the same ciphertext to the same plaintext (0s) with different keys.

Concrete example: AES-GCM

In AES-GCM, ciphertext block is encrypted by XOR’ing with the AES block cipher:

$$c_i = m_i + E_k(n + i).$$

Since the same ciphertext goes to the same plaintext

$$E_k(n + i) = E_k'(n' + i) \quad (1 \leq i \leq b).$$
Entering the Cryptography Territory...

- Hard to decrypt the same ciphertext to the same plaintext (0s) with different keys.
- AES-GCM: $E_k(n + i) = E_{k'}(n' + i)$ $(1 \leq i \leq b)$.
- If we model AES as an ideal cipher (no way to know the permutation without testing the key):

  \[
  E_k:
  \begin{array}{cccc}
  \vdots & \vdots & \vdots & \vdots \\
  0 & (n + 1) & (n + b) & 2^{128} - 1 \\
  \end{array}
  \]

  \[
  E_{k'}:
  \begin{array}{cccc}
  \vdots & \vdots & \vdots & \vdots \\
  0 & (n' + 1) & (n' + b) & 2^{128} - 1 \\
  \end{array}
  \]

Pretty hard to get $b$ 128-bit blocks to be the same!
Take 1: HTTPS

- Popular fix for key commitment: **Padding**
- Now let us look at HTTPS...
Take 1: HTTPS

- Popular fix for key commitment: **Padding**
- Now let us look at HTTPS...

```
HTTP/1.1 200 OK
Date: Wed, 24 Jul 2024 23:41:36 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
...
```

```
HTTP/1.1 200 OK
Date: Wed, 24 Jul 2024 23:47:49 GMT
Expires: Tue, 31 Mar 1981 05:00:00 GMT
Pragma: no-cache
...
```

*https://google.com*  *https://twitter.com*
Take 1: HTTPS

- Popular fix for key commitment: **Padding**
- Now let us look at HTTPS...

<table>
<thead>
<tr>
<th>HTTP/1.1 200 OK</th>
<th>HTTP/1.1 200 OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expires: -1</td>
<td>perf: 7402827104</td>
</tr>
<tr>
<td>Cache-Control: private, max-age=0</td>
<td>expiry: Tue, 31 Mar 1981 05:00:00 GMT</td>
</tr>
<tr>
<td>Content-Type: text/html; charset=UTF-8</td>
<td>pragma: no-cache</td>
</tr>
</tbody>
</table>

https://google.com  https://twitter.com

- It turns out that HTTPS is (kind of) padded!
■ It turns out that HTTPS is (kind of) padded!
■ Specifics vary, but most start with status code and date
  ■ also recommended by RFC 7231
It turns out that HTTPS is (kind of) padded!
Specifics vary, but most start with status code and date
  also recommended by RFC 7231
  If we consider all status codes (63) and the last hour (3600)...
  Only $63 \times 3600$ possibilities for the first 56 bytes!
  Define as variably padded
It turns out that HTTPS is (kind of) padded!
Specifics vary, but most start with status code and date
- also recommended by RFC 7231
- If we consider all status codes (63) and the last hour (3600)...
- Only $63 \times 3600$ possibilities for the first 56 bytes!
- Define as variably padded

We proved that proxy-based TLS is secure for HTTPS.
- Covers almost all websites!
Take 2: Non-HTTPS

Attacks incoming...

(1) Handshake and get key $k$

(2) **User derives plaintexts:** $k \rightarrow (0\text{dead}, 0\text{beef})$

(3) Please send me $0\text{dead}$

(4) Sends $0\text{dead}$

(5) **User somehow ‘proves’** $0\text{beef}$
Take 2: Non-HTTPS

- Attacks incoming...

  (1) Handshake and get key $k$

  (2) **User derives plaintexts**: $k \rightarrow (0x\text{dead}, 0xbeef)$

  (3) Please send me 0x\text{dead}

  (4) Sends 0x\text{dead}

  (5) **User somehow ‘proves’** 0xbeef

How likely will this attack happen?

Proxying is Enough: Security of Proxying in TLS Oracles and AEAD Context Unforgeability
Take 2: Non-HTTPS

Malleable

Account balance
Bank statement

... 

Insecure

Fixed

Account number
Age

... 

Insecure?
For fixed data, we need only a weaker key commitment property for the cipher suite.
- We define as context unforgeability (CFY).
- Informally: For fixed plaintext, hard to find another plaintext that matches the ciphertext
- Like second-preimage resistance in hash functions
<table>
<thead>
<tr>
<th><strong>Take 2: Non-HTTPS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AES-GCM</strong></td>
</tr>
<tr>
<td>AES is a block cipher (reversible).</td>
</tr>
<tr>
<td>Not secure under CFY</td>
</tr>
<tr>
<td><strong>Cannot</strong> be used in non-HTTPS scenarios</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td><strong>Chacha20-Poly1305</strong></td>
</tr>
<tr>
<td>Chacha20 is based on PRF (not reversible).</td>
</tr>
<tr>
<td>Secure under CFY</td>
</tr>
<tr>
<td><strong>Can</strong> be used in non-HTTPS scenarios with fixed data</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>
Proxy-Based TLS Oracles

HTTPS

Secure!
Almost all use case

Non-HTTPS

Secure?
Make sure data is fixed
Use Chacha20-Poly1305